Automatic conversion of XML-based contents into efficient executables for resource-limited environments

Imran A. Zualkernan, Yaser A. Ghanam, Amir S. Kalbasi, and Mohammed F. Shoshaa

izualkernan@aus.edu

American University of Sharjah, Sharjah, UAE

***Abstract* — There are two somewhat conflicting development trends in computer engineering. On one hand, there is a push towards massively parallel networks of severely resource-limited devices such as ad-hoc wireless networks. On the other hand, due to the emergence of web-services, resource-intensive XML has become a default representation for communication and processing. The problem is particularly acute in situations where XML needs to be processed on resource-limited devices such as mobile phones. This paper presents an architecture and a methodology for dynamically generating resource-optimized native run-time executables for such situations. A specific implementation based on this approach shows that size of the resulting executables is reduced by a factor of four when compared with the raw input XML files.**

***Index Terms* — Mobile Networks, Ad-hoc wireless networks, resource limited devices, Internet, XML.**

I. INTRODUCTION

Due to its support for interoperability and extensibility, for being the commonly accepted Meta-language in information technology, XML is being used in an increasingly broader array of computer applications. In many instances, XML is making it possible to implement "write once - run everywhere" applications where the content is represented in an XML format that is independent of the technology of the processing or rendering as shown in Fig. 1.

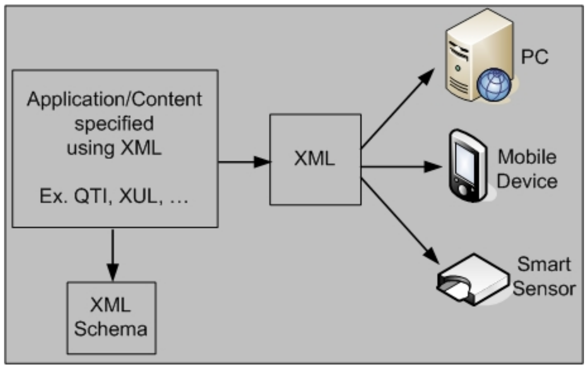

Figure 1. "write once – run everywhere" model based on XML

For example, Adobe Flex [1] is a presentation server to enable developers to develop multimedia applications using an XML-based language. Mozilla's XUL [2] which is another XML-based user interface language that enables building feature-rich XML-based cross-platform applications. Another instance is IMS QTI [3] which is an XML-based standard used widely in e-learning for authoring cross-platform assessments.

However, resource-limited environments such as mobile phones, PDA's and smart sensors, have severe limitations on power and memory resources which makes delivering, parsing, saving and processing XML-based contents on the device a difficult proposition. With the rapid emergence of mobile applications that depend heavily on XML, the need to develop an efficient approach to deliver and render XML contents on handheld devices is becoming critical.

This paper presents a generic approach to dynamically convert XML-based content into self-contained executables that consume significantly less power and size in resource-limited environments. An instance implementation of the proposed architecture is also discussed along with the results.

II. BACKGROUND

The problem of handheld devices being inefficiently used to process and render XML contents due to the limitations on CPU speed, volatile and non-volatile memory availability and screen size has been widely investigated in literature. For example, XMLZip [4] is an approach based on compression and decompression for XML documents based on the W3C DOM model. Other similar approaches include tools like XML-Xpress[5], Millau[6] and XMILL[7]. However, as [8] asserts, such tools and approaches require a considerable amount of RAM and CPU resources which makes them impractical for mobile devices. An alternative approach is suggested by XText [8] which is an efficient encoding and decoding scheme for XML that is fast and simple to implement.

Another example of efforts highlighting the parsing issue is kXML [9] which is a small XML pull parser intended for resource-limited environments to access, parse, and render XML files for J2ME-capable devices. Finally, [10] has tried to find properties in XML that regulate its high cost and tested "a hypothesis that tag-redundancy can be exploited to make parsing of XML cheaper." There are also existing commercial products such as Efficient XML [11] that propose solutions for the XML file size issue.

III. Approach

The approach discussed in this paper proposes a physical separation between processing and rendering of an XML document on a resource-limited device. In this approach, XML files or packages are processed along with any associated resources on the server side to produce an executable file that can be streamed on demand to a resource-limited device for rendering and processing. The resource-limited device only needs a small shell that receives the executable file and runs it.

As shown in Fig. 2, the server receives an XML package that has a collection of XML files accompanied with any required resources.

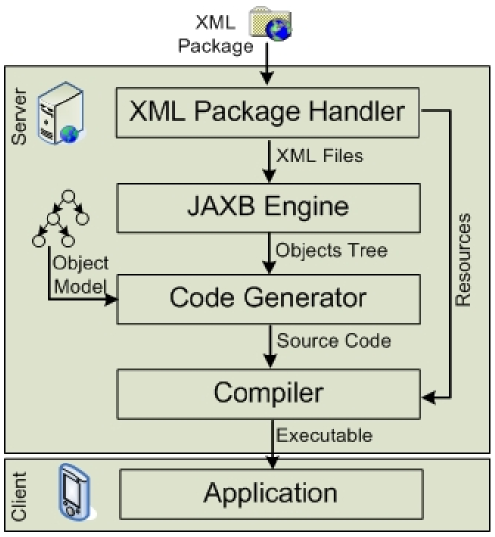

Figure. 2. System architecture for the general solution

The XML package handler processes the files within the package to separate the resources and direct the XML files toward the Java Architecture for XML Binding (JAXB) engine [12]. The JAXB engine produces a tree of java objects corresponding to the XML tags and attributes used in the package. The produced tree along with a specialized object model, which represents a hierarchy of classes used to structure the code generation process, are subsequently fed into the code generator, possibly an interpreter or a template engine.

The code generator constructs the source code as required for a specific run-time environment such as J2ME, .Net or Flash Lite [13]. The automatically generated source code is then compiled, and any required resources are bundled at this stage if possible. Alternatively, resources may be brought at a later stage upon the request of the shell in the handheld device.

The final output of the aforementioned process is a single file or a set of files that can be streamed on demand to the resource-limited device where the shell has to regulate the streaming and rendering process. Thus, the device does not have to parse or save any raw XML contents, which implies a substantial reduction of computational power and storage space on the device.

The next section presents a sample implementation of the suggested approach for handling a complex XML-based standard.

IV. Sample Application

QTI [3] is a popular XML-based standard for launching and rendering assessments over the internet. The full-fledged standard has over 400 different XML tags making the standard very flexible. However, processing all these tags has become increasingly expensive in resource-constrained devices.

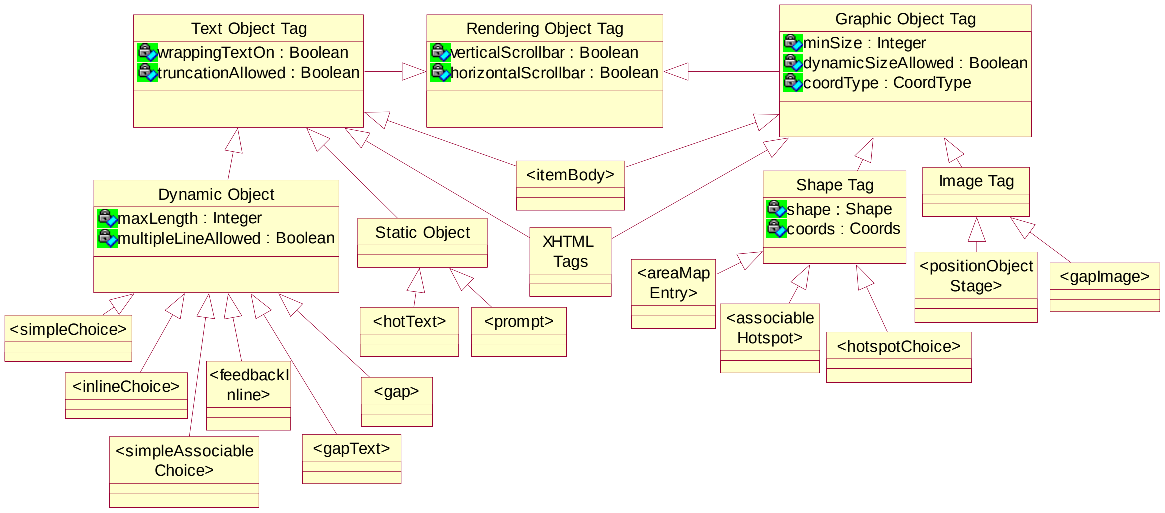

Figure 3. Object Model for mobile processing and rendition for QTIv2.1.

As a sample implementation of the proposed architecture, an engine was developed to convert QTIv2.1 packages in the XML format into an executable Flash Lite [13] file to be delivered and rendered on mobile devices. The process of producing an executable Flash Lite file for a given QTIv2.1 package was carried out according to the model previously shown in Fig. 2.

The QTIv2.1 assessment is first packaged as a group of XML files accompanied with the required resources. This package is fed into the code generation phase where JAXB produces java objects corresponding to the QTI files. After the system automatically recognizes the type of question under processing (e.g. Multiple Choice, True/False), the Velocity template engine [14] is employed to generate source code in Action Script 2.0. An object model developed especially for QTIv2.1, as partially shown in Fig. 3, is used to structure the code generation phase.

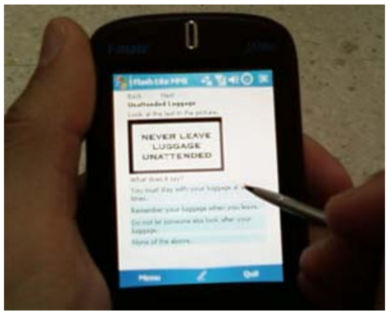

Figure 4. An example of an automatically generated Flash Lite application running on a mobile device.

Besides generating code for each assessment item, the Velocity engine also produces a shell that holds all the questions in the assessment, and manages navigation and response submission. In the end, MTASC [15] is used to compile the Actionscript 2.0 files created by Velocity to create a shockwave file (swf) that is ready to be streamed and rendered on the a resource-limited device as shown in Fig. 4.

V. Evaluation

One of the most important objectives of the architecture proposed is to reduce processing and the amount of data transmitted to the resource-limited device, and to thus save the storage space required. This section presents an evaluation of the sample architecture presented in the previous section.

To perform the analysis, 60 XML packages were randomly generated using a pool of XML components. Furthermore, the packages were used to generate the Action Script code for the 60 packages which were automatically compiled to Shockwave files (SWF) representing self-contained Flash Lite applications.

Fig. 5 shows the size of generated Shockwave files versus the size of the original XML packages. A linear regression fits the following equation ($R^2 = 0.8925$; $p < 0.001$) between SWF size and the size of the XML file (in KB).

$$SWF\_SIZE = 0.254 * XML\_SIZE + 4.8914 \quad (1)$$

This equation shows that the SWF size increases by 0.254 KB for every 1 KB increase in the XML file size. This implies a size reduction factor of 1/0.254 = 3.94 or about four times. The fixed size of the automatically generated empty shell is about 1.28 KB.

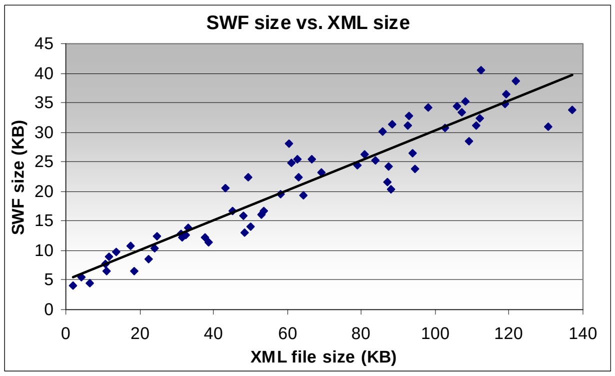

Figure 5. Size comparison of automatically generated SWF files vs. size of the original XML files.

The processing complexity of an XML file is approximately proportional to the number of nodes it has. Similarly, the processing complexity of a Flash Lite file is proportional to the number of the byte-code operators.

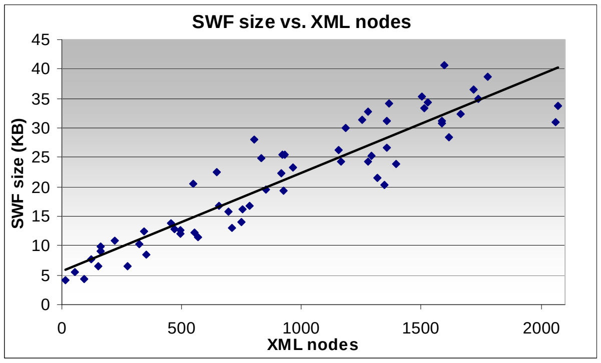

Figure 6. Size comparison of automatically generated SWF files vs. number of nodes in the original XML files.

Fig. 6. shows the relationship between the number of nodes in each XML file and the size (in KB) of the generated SWF file. A linear regression fits the following equation ($R^2 = 0.8526$ with $p < 0.001$).

SWF_SIZE = 0.0167 * NO_XML_NODES + 5.6897 (2)

This equation implies that the increase in the size of the SWF_SIZE is linear with respect to the number of nodes. In addition, the slope (node/KB) is small. This means that each additional node in the XML file adds about 16.7 bytes.

In the 60 XML packages analyzed, the average number of bytes per XML node was 70.79 (n=56,661). This means that from a XML-node perspective, there was a reduction of 70.79 bytes/node / 16.7 bytes/node = 4.24 times in the size.

VI. CONCLUSION

This paper presented an approach and architecture for dynamic conversion of XML-based contents into self-contained executables for resource-limited environments. The produced executables consume significantly less power and size than XML parsing engines on resource-limited devices. An instance of a possible application of the proposed architecture was discussed along with the resulted outcomes. For the specific example presented, the size of the transmitted contents was reduced by a factor of four. Moreover, the complexity of processing required on the mobile device was expected to decrease substantially. Future analysis will involve statistical study of the performance of the generated executables on handheld devices.

REFERENCES

[1] Flex Developer Center, http://www.adobe.com/devnet/flex/ , accessed April 30, 2007.

[2] The Joy of XUL, http://developer.mozilla.org/en/docs/The_Joy_of_XUL , accessed April 30,2007.

[3] IMS Question & Test Interoperability Specification, http://www.imsglobal.org/question/#version2.0, accessed Feb 10, 2007.

[4] D. Lenkov, “Binary XML”, http://www.w3.org/2003/08/binary-interchange-workshop/31-oracle-BinaryXML_pos.htm, accessed April 30, 2007.

[5] ICT, XML-Xpress, http://www.ictcompress.com/xml.html

[6] Girardot, M., and Sundaresan, N., “Millau: an encoding format for efficient representation and exchange of XML over the Web”, 9th International World Wide Web Conference, XML-Session 2.

[7] H. Liefke, and D. Suciu,, ?XMILL: An Efficient Compressor for XML Data. SIGMOD, 2000. http://www.research.att.com/sw/tools/xmill/

[8] D. A. Lee, “XML encoding techniques for storing XML data on memory limited (mobile) devices”, XML 2006 Conference.

[9] kXML, http://kxml.sourceforge.net/about.shtml , accessed April 30, 2007.

[10] A. Knudsen, “Cheaper parsing of XML on Mobile Devices”, www.idi.ntnu.no/grupper/su/fordypningsprosjekt-2003/fordypning2003-Andreas-Knudsen.pdf , accessed April 30, 2007.

[11] “Lightning-Fast Delivery of XML to More Devices in More Locations”, http://www.agiledelta.com/product_efx.html , accessed April 29,2007.

[12] Java Architecture for XML Binding (JAXB), http://java.sun.com/webservices/jaxb/, accessed Feb 10, 2007.

[13] Flash Lite, http://www.adobe.com/products/flashlite, accessed Feb 12, 2007.

[14] The Apache Velocity Project, http://velocity.apache.org/, accessed Feb 10, 2007.

[15] Motion-Twin ActionScript 2 Compiler, http://www.mtasc.org, accessed Feb 10, 2007.